Graph Neural Networks and Deep Reinforcement Learning in Job Scheduling*

Maroš Bratko[1], Thomas Seidelmann[2] and Martin Holeňa[1,3,4]

[1]Charles University in Prague
[2]Otto von Guericke University of Magdeburg
[3]Czech Technical University in Prague
[4]Czech Academy of Sciences

Abstract

Priority Dispatching Rules (PDRs) are greedy heuristic algorithms used to obtain approximate solutions to the NP-hard Job-shop Scheduling Problem (JSSP). The manual design of PDRs requires domain knowledge to achieve good performance, which varies with scenario properties and objectives. Recently, deep reinforcement learning has been used to automate the process of designing PDRs, where PDRs are formulated as a Markov Decision Process exploiting graph representation of JSSP, and a graph neural network (GNN) selects the operations to be dispatched. We experimentally compare five published models with source code publicly available on GitHub and our extensions of those models on three different variants of JSSP. Our experiments show that the choice of input features significantly affects the model's performance regardless of the GNN architecture. This suggests that the feature selection is essential for learning high-quality PDRs.

Keywords

job-shop scheduling, graph neural networks, deep reinforcement learning, markov decision process

1. Introduction

Job scheduling considers the allocation of jobs to resources with the goal of minimizing the makespan. Each job consists of a sequence of operations, and each resource can process only one operation at a time [1].

Generally, the Job-shop Scheduling Problem (JSSP) assumes all jobs to be known apriori and each operation can be allocated only to one given machine [1]. In a Flexible Job-shop Scheduling Problem (FJSP), each operation can be allocated to any of the machines from a given subset of machines [2]. A Dynamic Job-shop Scheduling Problem (DJSP), one of the more common variants, tackles the stochastic aspects of modern manufacturing, e.g., the arrival of new jobs during the execution of the schedule or uncertain processing times [3].

Due to the NP-hardness of these problems [4], numerous approaches and heuristics have been used over time to yield approximate solutions, e.g. genetic algorithms [5].

Priority Dispatching Rules (PDRs) [6] are a heuristic method widely used in scheduling systems. Designing a high quality PDR is usually a very time-consuming task requiring extensive domain knowledge. Deep reinforcement learning (DRL) has already been proposed as a possible solution for automatizing algorithm learning [7]. Several recent works have focused on extending this technique to job scheduling [8, 9, 10, 11], applying graph neural networks (GNNs) on a graph representation of job scheduling problems.

This article considers five models published in literature with source code publicly available on GitHub. Three of those models solve JSSP, and two of them solve FJSP. We present our extensions of JSSP models to DJSP and experimentally compare their performance on public benchmarks for JSSP and FJSP. To compare our DJSP extensions, we model the DJSP as a Poisson process and generate test instances from JSSP benchmarks.

2. Problem formulation

The Job-shop Scheduling Problem (JSSP) consists of a set of jobs $\mathcal{J}$ and a set of machines $\mathcal{M}$ [1]. Each job has an associated sequence of operations $O_{ij} \in \mathcal{O}$, which must be processed in the given order. Operation O_{ij} represents uninterrupted processing of job $J_i \in \mathcal{J}$ on machine $M_j \in \mathcal{M}$ with processing time p_{ij}. Each machine can process only one operation at a time. A schedule is a set of start times S_{ij} for each operation O_{ij} that satisfies these constraints. The completion times $C_{ij} = S_{ij} + p_{ij}$ denote the end of each operation. The JSSP solution is a schedule minimizing the total makespan $C_{\max} = \max_{i,j}\{C_{ij}\}$ [8].

Computational Intelligence and Data Mining - 11th international workshop

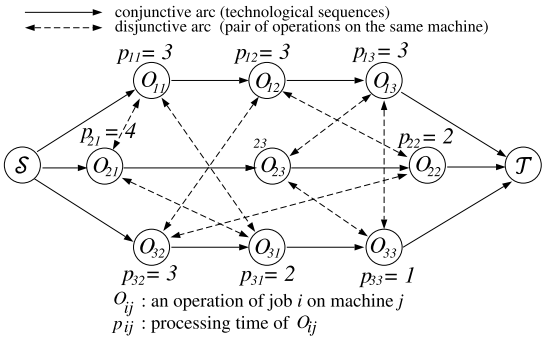

Figure 1: Disjunctive graph representation of JSSP, slightly modified from [1]

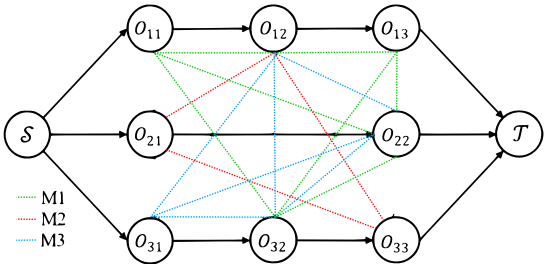

Figure 2: Disjunctive graph representation of FJSP, slightly modified from [14]

2.1. Flexible Job-shop Scheduling Problem

A flexible Job-shop Scheduling Problem (FJSP) is an extended version of JSSP with the only difference being that each operation $O_{ij} \in \mathcal{O}$ can be processed on any machine M_k from a given subset of machines $\mathcal{M}_{ij} \subseteq \mathcal{M}$ with processing time p_{ijk} [10]. Solving FJSP then consists of selecting the appropriate machine for each operation in addition to determining its start time.

2.2. Dynamic Job-shop Scheduling Problem

A dynamic Job-shop Scheduling Problem (DJSP) is a dynamic version of JSSP, where the jobs are not released all at once, but at different times throughout the execution. We assume that n jobs are known at the beginning of the schedule and n' jobs arrive after the start of the schedule [12, 13].

2.3. Disjunctive Graph Representation

JSSP can be represented as a disjunctive graph $G = (O, A, E)$, where O denotes a set of vertices corresponding to different operations O_{ij} weighted by the processing time together with a *start* node $\mathcal{S}$ and a *target* node $\mathcal{T}$ with processing time equal to zero, representing start and end of the schedule, respectively [1]. A is a set of arcs representing precedence constraints. $E = \bigcup_k E_k$ is a set of edges, where E_k is a clique connecting operations that require the same machine M_k for their execution. An example of a JSSP instance represented as a disjunctive graph is shown in Figure 1.

Finding a solution to the Job-shop Scheduling Problem can be viewed as defining the ordering between operations requiring the same machine. In the disjunctive graph, this is done by turning all edges into arcs in such a way that the resulting graph is a directed acyclic graph (DAG) [1].

2.3.1. FJSP as a disjunctive graph

The only difference with respect to JSSP is that each operation can be part of multiple cliques. An example of disjunctive graph representation for FJSP is shown in Figure 2.

2.4. Heterogeneous Graph Representation

JSSP as a heterogenous graph is $H = (O, M, E)$, where O is a set of operation nodes, M is a set of machines nodes, and E is a set of edges [11].

Each edge can be either operation-to-operation ($O-O$) edge, machine-to-machine ($M - M$) edge, or operation-to-machine ($O - M$) edge. $O - O$ edges fully connect all operations in the same job, and all machines are fully connected via $M - M$ edges. $O - M$ edge connects operations with machines on which they can be processed.

2.4.1. FJSP as a Heterogeneous Graph

FJSP as a heterogenous graph is defined as $H = (O, M, A, \Sigma)$ [10]. Set of operation nodes O and set of arcs A is the same as in the disjunctive graph. A set of machine nodes M represents machines, and a set of edges Σ connects operation nodes and machine nodes on which they can be processed. An example of a heterogeneous graph for FJSP is shown in Figure 3.

2.5. Priority Dispatching Rules

PDRs are a greedy heuristic method for solving JSSP in $|\mathcal{O}|$ steps [8]. For each eligible operation, PDR computes a priority index and selects the one with the highest priority to be dispatched. In FJSP, PDR also selects the machine. Traditional PDRs compute the priority index based on the set of features for each operation [13]. Traditional PDRs include:

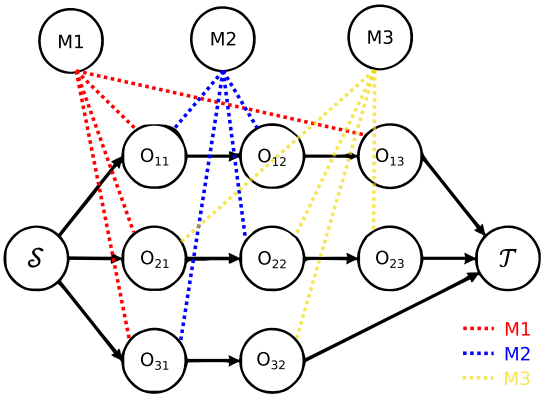

Figure 3: Heterogeneous graph representation of FJSP, slightly modified from [14]

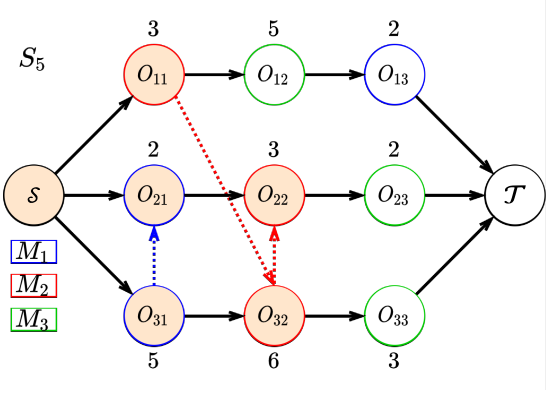

Figure 4: Disjunctive graph with original precedence constraints (black arrows), and three precedence constraints (red and blue arrows) between already scheduled operations (orange circles) in JSSP "arc adding scheme". Slightly modified from [8].

- First In First Out (FIFO)
- Most Operations Remaining (MOR)
- Shortest Processing Time (SPT)
- Most Work Remaining (MWKR)
- Earliest Due Date (EDD)
- Least Operations Remaining (LOR)
- Longest Processing Time (LPT)
- Least Slack (LS)
- Shortest Remaining Processing Time (SRPT)

Decisions made by PDRs can then be viewed as actions changing the disjunctive graph. This process can then be formulated as a Markov Decision Process (MDP) [8], allowing PDRs to be learned automatically via deep reinforcement learning techniques. State s_t at timestep t is a graph as described in Section 2.3 and Section 2.4. Action a_t at timestep t is an unscheduled operation whose preceding operations have already been scheduled. A state transition from s_t to s_{t+1} after executing a_t is done by updating the graph. The reward for each action is the difference between the $C_{\max}(s_t)$ and $C_{\max}(s_{t+1})$, where $C_{\max}(s_t)$ is the lower bound of the makespan in state s_t. The cumulative reward with discount factor $\gamma = 1$ is $C_{\max}(s_0) - C_{\max}(s_{|\mathcal{O}|})$, where $C_{\max}(s_{|\mathcal{O}|})$ corresponds to the makespan of the final schedule $C_{\max}$, and $C_{\max}(s_0)$ is a constant specific to the problem instance. Maximizing cumulative reward minimizes final schedule makespan $C_{\max}$ [8, 10, 11]. Each model presented in the next section uses their own modified version of this MDP, which we will not discuss in detail.

3. Models with source code

In this section, we will briefly present five models from the literature with available source code. These models used GNNs and DRL to solve JSSP and FJSP. We named each model after its GitHub repository. Source code of all models, experiments, and data used in this paper can be found on GitHub (link).

3.1. JSSP Models

L2D is a model published in [8], source code was obtained from [15]. It employs a modified disjunctive graph shown in Figure 4, where the authors start with the original disjunctive graph containing only conjunctive arcs, and after each dispatched operation, they add a conjunctive arc representing a new precedence constraint between operations on the same machine. To train the model, the authors used Proximal Policy Optimization (PPO) algorithm [16].

Wheatley is an open-source model published in [17], with source code available on GitHub [18]. It uses a similar "arc adding scheme" as **L2D**. To train the model, the authors used PPO algorithm [16].

IEEE-ICCE-RL-JSP was published in [11], the code was obtained from the GitHub repository [19]. It represents JSSP as a heterogeneous graph. In each step, this model chooses one of the traditional PDRs to determine the operation to dispatch. To train the model, the authors used a Double Deep-Q Network (DDQN) algorithm [20].

3.2. FJSP Models

End-to-end-DRL-for-FJSP was published in [14], the source code was obtained from [21]. It represents FJSP as

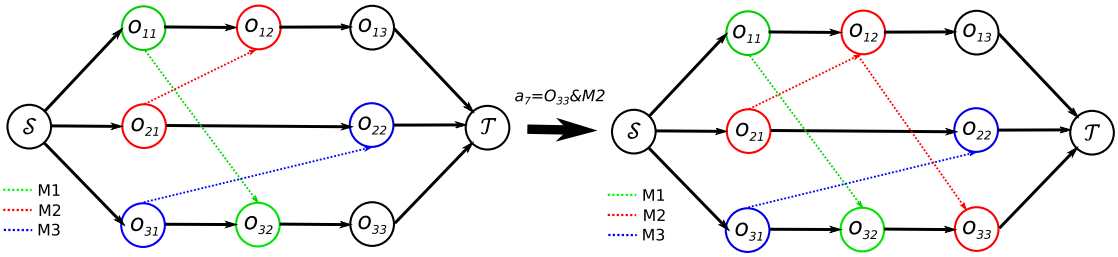

Figure 5: Action a_7 dispatches O_{33} on $M2$ by adding a red arc corresponding to $M2$ and changing the color of O_{33} to red in FJSP "arc adding scheme". Reproduced from [14].

a disjunctive graph. It uses a similar *"adding arc scheme"* as **L2D**, which is shown in Figure 5 for an FJSP. To train the agent, the authors used a multi-agent version of the PPO algorithm.

The model **fjsp-drl** was published in [10], and the source code was obtained from [22]. It represents FJSP as a heterogeneous graph. To train the model, the authors used their own modified version of PPO algorithm.

4. Proposed Extensions

In this section, we will present our extensions of JSSP models to DJSP, namely for **L2D**, **Wheatley**, and **IEEE-ICCE-RL-JSP**.

4.1. Dynamic L2D

The main idea behind our algorithm is that when a new job J_{new} arrives at the time t_a, the algorithm reschedules all operations that have not started yet. The complete algorithm pseudocode is shown in Algorithm 1. We treat L2D as a black box, which takes a JSSP instance and a list of actions as input (containing the already started operations), performs actions from the list in a given order, dispatches the remaining operations using its GNN, and outputs a solution. The already started operations, and their start times, are kept in a list called *plan*. After each new job arrival, plan is updated with operations that have started since the previous job arrived, the new job is added to the instance, and L2D is used to dispatch the rest of the operations.

Algorithm 1 Dynamic **L2D**

Input: Jobs known at the start J_{known}, $queue$ with arriving jobs
Output: Dynamic schedule

$plan \leftarrow$ initialize empty list of actions
$instance \leftarrow$ formulate JSSP instance from J_{known}
$solution \leftarrow$ dispatch operations in $instance$ with **L2D**
while new jobs are expected to arrive **do**
  if no new job arrived in $queue$ **then**
    continue
  end if
  $J_{\text{new}} \leftarrow$ new job from the $queue$
  $t_a \leftarrow$ arrival time of the job J_{new}
  for operation O_{ij} in $solution$ **do**
    if $S_{ij} < t_a$ and O_{ij} not in $plan$ **then**
      add O_{ij} to the $plan$
    end if
  end for
  add J_{new} to J_{known}
  $instance \leftarrow$ create JSSP instance from J_{known}
  for operation O_{ij} in $plan$ **do**
    dispatch O_{ij} in $instance$
  end for
  $solution \leftarrow$ dispatch the rest of $instance$ with **L2D** and constraint $S_{ij} \geq t_a$
end while
return $solution$

4.2. Dynamic Wheatley

We consulted Wheatley's authors in the GitHub issue [23]. We were advised to add a "virtual" operation at the start of the job processed by a "virtual" machine with the processing time equal to the arrival time t_a. This "virtual" operation guarantees, that the first "real" operation will start after t_a. To avoid different jobs interfering with each other, we add one "virtual" machine per one "virtual" operation.

4.3. Dynamic IEEE-ICCE-RL-JSP

We modified **IEEE-ICCE-RL-JSP** to take a list of jobs and their arrival times as input. When the model chooses which operation to dispatch, operations from jobs that have not yet arrived are not listed as eligible operations.

5. Experiment

In this section, we will experimentally compare the models presented in Section 3 and Section 4. We will describe the experimental setup for each job scheduling variant (JSSP, FJSP, DJSP) and interpret the results.

5.1. Setup

We ran all experiments on a MacBook Pro with an Apple M2 Max chip, 32 GB of RAM, and Sonoma 14.4.1 macOS operating system.

5.1.1. Instances

JSSP. We obtained 242 benchmark instances, and their best-observed solutions from [24]. We reformulate each JSSP instance as its equivalent FJSP instance with $|\mathcal{M}_{ij}| = 1$, so FJSP models can process JSSP instances. For each model and JSSP instance, we run the experiment with 10 different seeds. We are interested in the *gap* given by the following equation

$$gap = \frac{C - C_{\text{best}}}{C_{\text{best}}}\,, \tag{1}$$

where C is the makespan produced by the model, and C_{best} is the best-observed makespan.

FJSP. We obtained 402 benchmark instances, and their best-observed solutions from [25]. Again, we repeat the experiment for each model and each FJSP instance with 10 different seeds and calculate the gap.

DJSP. We designed an experiment inspired by [26]. We assume a set of known jobs at the beginning. We then model the arrival of new jobs as a Poisson process, i.e., the arrival of two consecutive jobs follows an exponential distribution [26], where the average arrival time is

$$\Delta t_{\text{avg}} = \frac{\mu_a}{U}\,, \tag{2}$$

where μ_a is the average processing time of all operations, and U is a load factor of the dynamic job shop. We use $U \in \{1, 4\}$ in the experiment.

We generate the DJSP instances from the JSSP instances and consider half the jobs as known and the rest as arriving jobs. We repeat each experiment with 10 different seeds. We are interested in the makespan because the gap is not available.

5.1.2. Model checkpoints

L2D. From the **L2D** GitHub repository [15], we obtained checkpoints trained on random instances with size 6x6, 10x10, 15x15, 20x15, 20x20, 30x15, 30x20.

Wheatley. For the experiment, we had to train **Wheatley** ourselves. To make **Wheatley** as similar to **L2D** as possible, we chose the same training hyperparameters as **L2D**.

The listed checkpoints were trained using a CPU on an Arm-based Ampere A1 virtual machine with 4 CPUs and 24 GB of RAM in Oracle Cloud Infrastructure with Ubuntu 20.04 64-bit operating system. We stopped the training when the model had not improved its objective for at least two days. Training of each checkpoint took at least 3 weeks.

IEEE-ICCE-RL-JSP. We used the provided training script, which trains the model until the *gap* is smaller than 20%. We trained five checkpoints. We trained the model using CPU on an Arm-based Ampere A1 virtual machine with 4 CPUs and 24 GB of RAM in Oracle Cloud Infrastructure with Ubuntu 20.04 64-bit operating system.

End-to-end-DRL-for-FJSP. We obtained only one checkpoint from the GitHub repository [21].

fjsp-drl. We obtained five checkpoints from the GitHub repository for this model [22].

5.1.3. Baselines

JSSP. We compared the models with the 9 classic PDRs listed in Section 2.5. We used implementations given by the **IEEE-ICCE-RL-JSP** model as obtained from the GitHub repository [19].

FJSP. For FJSP, we used PDR implementations from the code of the model **End-to-end-DRL-for-FJSP**. For operation selection, we used FIFO, MOPNR, LWKR, and MWKR. For machine selection, we used EET and SPT.

5.2. Results

5.2.1. JSSP

Average gaps are shown in Table 1. The boxplot of JSSP gaps is shown in Figure 6. Average runtimes are in Table 2.

We rejected the null hypothesis that the medians of the performance of the three best models (**MWKR**, **fjsp-drl**, and **IEEE-ICCE-RL-JSP**) are equal using the Kruskal-Wallist test [27] ($p < 5\%$). Therefore we can also reject the null hypothesis that the medians of the performance of all models are equal.

We compared all models pairwise using the Mann-Whitney U test [28] corrected for multiple hypotheses testing using the Holm method [29]. The best three models were significantly different from the rest. The pairwise comparison of the three best models is shown in

Table 1
Average JSSP gaps from optimal solutions for different models; the lowest average value is in **bold**

Model	Gap [%]
EDD	0.36 ± 0.12
FIFO	0.30 ± 0.13
LOR	0.40 ± 0.11
LPT	0.44 ± 0.11
MWKR	0.22 ± 0.12
LS	0.29 ± 0.11
MOR	0.26 ± 0.14
SPT	0.26 ± 0.09
SRPT	0.40 ± 0.11
End-to-end-DRL-for-FJSP	0.30 ± 0.14
fjsp-drl	0.22 ± 0.12
IEEE-ICCE-RL-JSP	**0.20 ± 0.12**
L2D	0.31 ± 0.16
Wheatley	0.48 ± 0.38

Table 2
Average JSSP runtimes for different models and categories

Model	Runtime [s]
EDD	4.04 ± 6.11
FIFO	1.69 ± 3.51
LOR	1.09 ± 2.26
LPT	2.20 ± 3.39
MWKR	2.31 ± 3.82
LS	2.43 ± 3.88
MOR	1.26 ± 2.58
SPT	2.10 ± 3.37
SRPT	2.41 ± 3.82
End-to-end-DRL-for-FJSP	4.97 ± 8.69
fjsp-drl	16.70 ± 38.10
IEEE-ICCE-RL-JSP	9.85 ± 12.17
L2D	2.46 ± 3.18
Wheatley	10.98 ± 24.60

Table 3
Achieved significancies (p-values) of the comparison between the three best models JSSP models, Holm-corrected for the simultaneous testing of all 91 considered pairwise hypotheses

Comparison	Corrected p-value
IEEE-ICCE-RL-JSP vs. fjsp-drl	8.1%
fjsp-drl vs. MWKR	100%
IEEE-ICCE-RL-JSP vs. MWKR	3.3%

Table 3. The best model is not significantly different from the second, the second best model is not significantly different from the third, but the best model *is* significantly different from the third.

The Holm-corrected p-values of pairwise comparison using the Mann-Whitney U test are shown in Table 3.

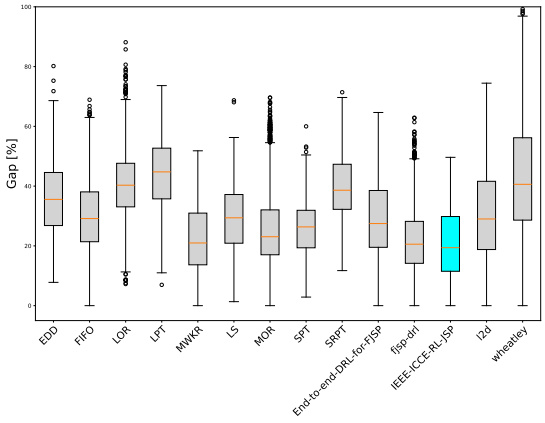

Figure 6: Boxplot of JSSP gaps from optimal solutions; The model with the lowest average gap is highlighted as a blue box

Table 4
Average FJSP gaps from optimal solutions for different models and instance categories; the lowest values are in **bold**

Model	Gap [%]
FIFO-EET	0.67 ± 1.13
FIFO-SPT	0.37 ± 0.22
LWKR-EET	1.10 ± 1.10
LWKR-SPT	0.64 ± 0.32
MOPNR-EET	0.97 ± 1.35
MOPNR-SPT	0.40 ± 0.22
MWKR-EET	0.80 ± 1.13
MWKR-SPT	0.40 ± 0.21
End-to-end-DRL-for-FJSP	**0.13 ± 0.13**
fjsp-drl	0.21 ± 0.35

5.2.2. FJSP

Average FJSP gaps for different models are shown in Table 4. Average runtimes are shown in Table 5. A boxplot for FJSP gaps is shown in Figure 7 with logarithmic vertical scale.

We rejected the null hypothesis that the medians of gaps of all models and PDRs are equal using the Kruskal-Wallis test ($p < 5\%$). We also compared models and PDRs using the Holm-corrected Mann-Whitney U test [28]. Almost all pairwise comparisons produced statistically significant differences ($p < 5\%$). Comparisons that did not produce statistically significant differences are **MOPNR-SPT** vs. **MWKR-EET**, **MOPNR-SPT** vs. **MWKR-SPT**, and **MWKR-EET** vs. **MWKR-SPT**.

Table 5
Average FJSP runtimes for different models

Model	Runtime [s]
FIFO-EET	0.13 ± 0.24
FIFO-SPT	0.13 ± 0.23
LWKR-EET	0.12 ± 0.20
LWKR-SPT	0.12 ± 0.20
MOPNR-EET	0.13 ± 0.21
MOPNR-SPT	0.13 ± 0.22
MWKR-EET	0.12 ± 0.21
MWKR-SPT	0.13 ± 0.21
End-to-end-DRL-for-FJSP	1.13 ± 1.63
fjsp-drl	1.70 ± 2.27

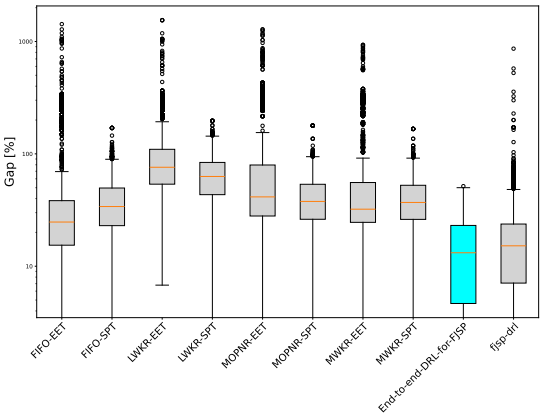

Figure 7: Boxplot of FJSP gaps from optimal solutions with logarithmic vertical scale; The model with the lowest average gap is highlighted as a blue box

5.2.3. DJSP

Average makespans with the load factor 1 are shown in Table 6, with the load factor 4 in Table 7. We tested the null hypothesis for each size and load factor that the medians of the performance of all models are equal using the Holm-corrected Kruskal-Wallis U test, and after we rejected this null hypothesis, we compared the models pairwise using the Holm-corrected Mann-Whitney U test.

6. Discussion

6.1. JSSP

The results for **MWKR** we obtained are much better than the results reported by the authors of **L2D** [8]. We did not find their implementation of **MWKR**, so we could not investigate the difference.

Table 6
Average DJSP makespan with $U = 1$; the lowest values are **bold**; "**" denotes best model better than the other two models, "*" denotes best model better only than the last model or the second best model better than the last model

Size	L2D	IEEE-ICCE-RL-JSP	Wheatley
6x6	**72.03 ± 9.40**	74.69 ± 11.38	77.00 ± 9.24
10x5	**780.24 ± 90.34***	874.43 ± 153.79	817.90 ± 78.99
10x10	**1204.43 ± 219.98***	1259.91 ± 257.09	1212.22 ± 217.04
15x5	**1031.16 ± 61.63****	1198.84 ± 108.35	1080.24 ± 89.20*
15x10	**1329.93 ± 94.21****	1413.15 ± 116.03	1442.65 ± 129.44
15x15	**1676.67 ± 116.37****	1716.56 ± 119.06*	1817.81 ± 145.40
20x5	**1363.52 ± 148.91***	1701.57 ± 203.56	1367.97 ± 104.40*
20x10	**1858.25 ± 213.24***	2081.56 ± 257.26	1918.04 ± 161.88*
20x15	**2779.05 ± 1270.01****	3009.12 ± 1383.13	2988.77 ± 1338.80
20x20	**3208.98 ± 1522.30****	3402.65 ± 1637.48*	3655.56 ± 1662.28
30x10	**2143.03 ± 126.54****	2511.60 ± 222.45	2510.66 ± 181.78
30x15	**4034.07 ± 1660.43****	4484.68 ± 1833.57*	4819.48 ± 1847.26
30x20	**4469.95 ± 1779.81****	4844.21 ± 1961.36*	5805.12 ± 2149.98
40x15	**6835.81 ± 1080.76****	7789.19 ± 1137.65*	8884.73 ± 862.80
40x20	**7556.13 ± 1071.72****	8496.36 ± 1179.83*	10985.64 ± 1032.40
50x10	**3809.08 ± 766.24****	4456.46 ± 684.34*	4948.68 ± 604.10
50x15	**5882.77 ± 2621.21****	6689.03 ± 2736.00*	9118.45 ± 3524.75
50x20	**6320.83 ± 2770.31****	7050.68 ± 3024.71*	11367.83 ± 4334.55
100x20	**6310.22 ± 178.01****	8150.22 ± 347.98*	22838.04 ± 1550.05

Table 7
Average DJSP makespan with $U = 4$; the lowest values are **bold**; "**" denotes best model better than the other two models, "*" denotes best model better only than the last model or the second best model better than the last model

Size	L2D	IEEE-ICCE-RL-JSP	Wheatley
6x6	70.05 ± 8.25	**64.48 ± 3.84****	75.50 ± 8.55
10x5	747.63 ± 75.82	**741.80 ± 78.28**	775.62 ± 77.70
10x10	1176.08 ± 221.28	**1109.18 ± 200.95****	1187.33 ± 212.06
15x5	1006.18 ± 53.37	**984.55 ± 44.94***	1008.57 ± 61.81
15x10	1274.83 ± 76.31*	**1191.63 ± 70.89****	1321.84 ± 85.92
15x15	1598.28 ± 96.13*	**1506.20 ± 70.61****	1681.24 ± 97.81
20x5	1343.42 ± 184.64	1339.17 ± 134.68	**1333.13 ± 131.26**
20x10	1835.01 ± 272.46	**1710.64 ± 218.77****	1868.15 ± 233.91
20x15	2695.64 ± 1268.49*	**2541.11 ± 1179.87****	2852.97 ± 1325.87
20x20	3066.77 ± 1509.04*	**2902.33 ± 1419.60****	3446.61 ± 1688.14
30x10	2036.46 ± 114.04*	**1993.52 ± 92.45****	2207.02 ± 132.10
30x15	3955.84 ± 1730.05*	**3742.95 ± 1567.90****	4412.12 ± 1870.05
30x20	4330.06 ± 1818.44*	**4122.12 ± 1719.04****	5214.83 ± 2106.46
40x15	6757.92 ± 1322.93*	**6408.27 ± 1070.14****	7954.02 ± 1196.00
40x20	7463.87 ± 1319.68*	**7114.58 ± 1091.71****	9642.57 ± 1169.56
50x10	3885.63 ± 873.73*	**3676.22 ± 659.45****	4241.08 ± 729.50
50x15	5810.45 ± 2723.58*	**5504.96 ± 2345.68****	7504.37 ± 3180.44
50x20	6227.76 ± 2894.36*	**5883.56 ± 2631.55****	9320.83 ± 3888.32
100x20	**6076.26 ± 226.25****	6225.12 ± 206.62*	16734.28 ± 1051.45

Good performance of **MWKR** hints at the importance of the sum of processing times of remaining operations as an operation feature. The model **IEEE-ICCE-RL-JSP** used the number of remaining operations in the current job as an operation feature. The model **fjsp-drl** also included the number of unscheduled operations in the job as a feature. Other models didn't use this feature.

6.2. FJSP

The machine selection PDR **SPT** consistently performs significantly better than **EET**. The model **End-to-end-DRL-for-FJSP** includes processing time as a feature ex-

plicitly in the machine embeddings. The model **fjsp-drl** includes the time when the machine will finish all its currently assigned operations as a feature which is similar to the machine selection PDR **EET**. This may explain why the model **fjsp-drl** yielded worse results.

6.3. DJSP

The worse results yielded by **Wheatley** for all load factors can be a consequence of the poor performance in solving static JSSP.

The result of this experiment indicates that for more sparsely arriving jobs, **L2D** features (the lower bound of the estimated completion time) are more important. As the load factor increases, a DJSP is more similar to a static JSSP, and other features, which are more important for a JSSP, become more important, too.

7. Conclusion and future work

In this paper, we compared five job scheduling models with available source code. We extended three of those models to the dynamic variant of job-shop scheduling. From our experiments, we observed that the selection of input features had a much more significant impact on model performance than the architecture of the model. In JSSP, the remaining work seemed to be the most crucial. In FJSP, it was processing time during machine selection. In DJSP, the lower bound of the estimated completion time seemed more influential for sparsely incoming jobs. For more densely incoming jobs, the remaining work seemed to be more relevant. Investigating the effect of existing input features and searching for new ones may be a valuable direction for future work.

References

[1] T. Yamada, R. Nakano, Job shop scheduling, IEE control Engineering series (1997) 134–134.

[2] S. Dauzère-Pérès, J. Ding, L. Shen, K. Tamssaouet, The flexible job shop scheduling problem: A review, European Journal of Operational Research 314 (2024) 409–432. URL: https://www.sciencedirect.com/science/article/pii/S037722172300382X. doi:https://doi.org/10.1016/j.ejor.2023.05.017.

[3] J. Mohan, K. Lanka, A. N. Rao, A review of dynamic job shop scheduling techniques, Procedia Manufacturing 30 (2019) 34–39. URL: https://www.sciencedirect.com/science/article/pii/S2351978919300368. doi:https://doi.org/10.1016/j.promfg.2019.02.006, digital Manufacturing Transforming Industry Towards Sustainable Growth.

[4] M. R. Garey, D. S. Johnson, R. Sethi, The complexity of flowshop and jobshop scheduling, Math. Oper. Res. 1 (1976) 117–129. URL: https://api.semanticscholar.org/CorpusID:207233771.

[5] G. Zhang, L. Gao, Y. Shi, An effective genetic algorithm for the flexible job-shop scheduling problem, Expert Systems with Applications 38 (2011) 3563–3573.

[6] R. Haupt, A survey of priority rule-based scheduling, Operations-Research-Spektrum 11 (1989) 3–16. URL: https://api.semanticscholar.org/CorpusID:60820532.

[7] Y. Bengio, A. Lodi, A. Prouvost, Machine learning for combinatorial optimization: a methodological tour d'horizon, 2020. arXiv:1811.06128.

[8] C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, C. Xu, Learning to dispatch for job shop scheduling via deep reinforcement learning, 2020. arXiv:2010.12367.

[9] K. Lei, P. Guo, Y. Wang, J. Zhang, X. Meng, L. Qian, Large-scale dynamic scheduling for flexible job-shop with random arrivals of new jobs by hierarchical reinforcement learning, IEEE Transactions on Industrial Informatics 20 (2024) 1007–1018. doi:10.1109/TII.2023.3272661.

[10] W. Song, X. Chen, Q. Li, Z. Cao, Flexible job-shop scheduling via graph neural network and deep reinforcement learning, IEEE Transactions on Industrial Informatics 19 (2023) 1600–1610. doi:10.1109/TII.2022.3189725.

[11] K.-H. Ho, J.-H. Wu, F. Chiang, Y.-Y. Wu, S.-I. Chen, T. Kuo, F.-J. Wang, I.-C. Wu, Deep reinforcement learning based on graph neural networks for job-shop scheduling, in: 2023 International Conference on Consumer Electronics - Taiwan (ICCE-Taiwan), 2023, pp. 805–806. doi:10.1109/ICCE-Taiwan58799.2023.10226873.

[12] N. Kundakcı, O. Kulak, Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem, Computers & Industrial Engineering 96 (2016) 31–51. doi:https://doi.org/10.1016/j.cie.2016.03.011.

[13] R. Haupt, A survey of priority rule-based scheduling, OR Spektrum 11 (1989) 3–16. doi:10.1007/bf01721162.

[14] K. Lei, P. Guo, W. Zhao, Y. Wang, L. Qian, X. Meng, L. Tang, A multi-action deep reinforcement learning framework for flexible job-shop scheduling problem, Expert Systems with Applications 205 (2022) 117796. doi:https://doi.org/10.1016/j.eswa.2022.117796.

[15] C. Zhang, L2d, https://github.com/zcaicaros/L2D, 2020. Accessed 2024-03-17.

[16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms,

CoRR abs/1707.06347 (2017). URL: http://arxiv.org/abs/1707.06347. arXiv:1707.06347.

[17] G. Infantes, S. Roussel, P. Pereira, A. Jacquet, E. Benazera, Learning to solve job shop scheduling under uncertainty, in: B. Dilkina (Ed.), CPAIOR 2024: Integration of Constraint Programming, Artificial Intelligence and Operations Research (20th International Conference), Springer Nature, 2024.

[18] jolibrain, wheatley, https://github.com/jolibrain/wheatley/, 2024. Accessed 2024-01-09.

[19] Jerry-Github-Cloud, Ieee-icce-rl-jsp, https://github.com/Jerry-Github-Cloud/IEEE-ICCE-RL-JSP, 2023. Accessed 2024-03-17.

[20] H. van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, Proceedings of the AAAI Conference on Artificial Intelligence 30 (2016). URL: https://ojs.aaai.org/index.php/AAAI/article/view/10295. doi:10.1609/aaai.v30i1.10295.

[21] L. Kun, End-to-end-drl-for-fjsp, https://github.com/Lei-Kun/End-to-end-DRL-for-FJSP, 2022. Accessed 2024-03-17.

[22] W. Song, fjsp-drl, https://github.com/songwenas12/fjsp-drl, 2023. Accessed 2024-03-22.

[23] jolibrain, wheatley - different arrival times feature request, https://github.com/jolibrain/wheatley/issues/89, 2024. Accessed 2024-01-09.

[24] Job shop instances and solutions, http://jobshop.jjvh.nl/index.php, 2024. Accessed: 2024-03-28.

[25] D. Behnke, M. J. Geiger, Test instances for the flexible job shop scheduling problem with work centers (2012). doi:10.24405/436.

[26] K. Lei, P. Guo, Y. Wang, J. Zhang, X. Meng, L. Qian, Large-scale dynamic scheduling for flexible job-shop with random arrivals of new jobs by hierarchical reinforcement learning, IEEE Transactions on Industrial Informatics 20 (2024) 1007–1018. doi:10.1109/TII.2023.3272661.

[27] W. H. Kruskal, W. A. Wallis, Use of ranks in one-criterion variance analysis, Journal of the American Statistical Association 47 (1952) 583–621. doi:10.1080/01621459.1952.10483441.

[28] H. B. Mann, D. R. Whitney, On a test of whether one of two random variables is stochastically larger than the other, The annals of mathematical statistics (1947) 50–60.

[29] S. Garcia, F. Herrera, An extension on" statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons., Journal of machine learning research 9 (2008).